Autonomous Neural Network Controllers for Adaptive Material Handling

ONR Contract No. N00014-91-C-0258
Final Report
July 30, 1993

Dr. James Kottas
Dr. Michael Kuperstein
Symbus Technology, Inc.
1601 Trapelo Road
Waltham, MA 02154

Summary

For robots to be more useful in flexible manufacturing and service applications, the controllers must be able to handle more variable environments. On at least two levels, conventional methods in robot control have problems dealing with high variability. At the movement level, conventional dynamic control formulations cannot deal effectively with the highly variable dynamic inertial interactions between multijointed robots and payloads. At the task level, the initial and final positions for materials to be moved may change slightly but unexpectedly. We have developed autonomous neural network controllers that learn from their own experience to deal with environmental variability at these levels.

Our dynamic multijoint neural controller allows robots to learn to move diverse payloads from point to point without any knowledge of the robot structure or the gravitational environment. As tested on a PUMA 260 robot that is directly-controlled by a PC-based host, the performance of our controller in terms of final end-point accuracy and stability exceeds that of conventional dynamic control methods and is comparable to recently developed algorithms that rely on a model of the robot. Using sample payloads throughout the robot's range, the controller performs several movements and automatically updates its control parameters after each movement.

For task level variability, we consider a more specific materials handling application that current controllers cannot handle well: part insertion with unconstrained alignment. The main difficulty is that the target hole for a part may not be in the expected position and/or orientation. Passive compliant devices, currently available, provide only limited amounts of compliance and are part specific. Using a six degree-of-freedom force/torque sensor for feedback on the wrist of our PUMA 260 robot, we developed an initial prototype of a neural network controller which learns to put a peg into a hole using its own experience. It offers active compliance using the entire robot arm and allows greater variability for the target hole position and orientation. Using a video tape demonstration, Symbus Technology is actively pursuing customers that will help bring this technology to market.

Table of Contents

1. Introduction

1.1 The General Materials Handling Problem: Environmental Variability

Most applications of robots are in highly constrained manufacturing environments where there is a low degree of variability about where the materials are and where they are to be placed. To realize this tightly-controlled environment, expensive customized tooling currently must be used. When the manufacturing line changes, the retooling costs can be 80% of the total conversion cost. As a result, there is a trend toward flexible manufacturing environments whereby more capable robot controllers are needed to allow more variability in the workcell and thus reduce the amount of customized tooling required for a particular application.

The issue of environmental variability is even more critical in the service robot industry. A service robot would be handling materials in an environment that either cannot be modified at all or only at great expense. Since a service robot would most likely be sharing this environment with other active elements (such as people), the environment is highly variable with respect to the service robot because it could change unexpectedly over time. For example, a fallen box could block its path. A bin that the robot was supposed to fill with some material is still full from the previous day because it was not emptied as expected. Thus the need for flexible controllers that can deal effectively with varying environments is crucial for service robots.

1.2 Levels of Variability

With respect to the robot controller, the environmental variability can exist at several different levels. At the lowest level, dynamic control issues are relevant. Simple movements can be done with different payloads and varying speeds over time. The inertial qualities of the payload can interact with the inertial dynamics of the multijoint robot arm to produce oscillations at the end point of a movement, particularly at high speeds. A *stable* movement has no end-point oscillations. Furthermore, the movement is *accurate* if the robot is stationary at the desired place at the desired time. Over the robot's lifetime, friction in its joints will change, affecting the arm's inertial properties and thus a movement's end-point stability. If another robot of the same type is used in place of the original robot, both the accuracy and stability of the movements can be affected because no two robots are exactly the same. At the robot level, variabilities in the robot and its payload over time can result in poor movements due to the interaction of inertial dynamics.

At a higher level, a task which consists of several movements has sources of variability that are external to the robot. Consider the assembly line task of part insertion. If either the part or its target position is not in the expected position, the controller must compensate by generating a proper corrective movement. The main problem for the controller in this case is establishing and

maintaining calibration so that a sensed force/torque measurement of the new part position is translated into the appropriate corrective movement.

1.3 Conventional Approaches to Dealing with Variability

Primarily two methods are used to deal with variability. The first method is to develop detailed models of the process and how it might vary. The second method is to operate the process in a less-than-optimal regime. The manner in which these two methods are implemented differs between the robot and task levels.

At the robot level, detailed models of the robot kinematics and dynamics are used to control the movement of the robot. However, due to variations between robots of the same type, robot-specific calibration data must be provided for each robot. Because of the limitations of conventional control algorithms, the robots are larger, heavier, and slower than they need to be in order to reduce dynamic instabilities at the end-points of movements throughout the robot's movement range. Larger robots need more space and power to operate, thus increasing their cost to run. Furthermore, slower robots reduce process throughput.

At the higher task level, variability is reduced by having customized tooling to prevent parts and their destinations from having variable positions. Part-specific feeders are also used to constrain part presentation. However, these hardware methods are expensive to implement, particularly if the parts will be changing over time. For variability problems with part insertion, primarily two work-around methods are employed. In one method, if a part fails to be inserted on the first attempt, it is jiggled randomly for a preset amount of time to see if it will fall into place. If it doesn't, the part is discarded and a new one is tried. Alternatively, passive compliant devices can be used to hold the part. Although these devices are not expensive, their usefulness is limited in the amount of compliance they can provide and by the fact that they are still part specific.

1.4 Our Approach: Autonomous Neural Network Controllers

By contrast, our approach is to design controllers that do not require any models of the robots or their environments. They are based on neural network technology and learn using their own experience. Besides being able to handle variable environments more easily, this approach has the additional advantage in that our neural network controllers can be used in other non-robotic control applications which have the same inertial and variability issues, such as controlling temperature and fluid flow.

After an initial training period, our neural network controllers can be put on-line while being trained continuously to maintain calibration. In effect, the learning process generates a set of robot-

specific customization data that can evolve as the environment changes. In this way, our neural network controllers are autonomous. In order to adapt to new conditions, they only require the ability to sense that new conditions are present. Examples of new conditions are increased friction in the joints of a robot (which can be sensed by a reduction in the expected speed of a known movement) and a shifted pick-up location for a part (which can be sensed by a camera or proximity sensors). The neural network controllers can accept any type of sensor input.

In this project, we developed two autonomous neural network controllers for two levels of variability: robot and task. At the robot level, our dynamic multijoint neural controller can learn to move a payload from one position to another (point-to-point) with a high degree of stability and accuracy. By incorporating a sense of the payload to be moved, the controller can handle novel payloads with every movement. For overall dynamic stability of the robot, the controller is designed as a feedforward computed-torque formulation with simple position and velocity feedback loops. Both the feedforward and feedback paths have adaptable governing parameters which are fixed for any particular movement. These parameters are stored by the controller's neural network. At the end of a movement, the stability and accuracy are sensed and an error signal is generated to adapt the neural network so as to adjust these parameters in the appropriate way. The dynamic multijoint neural controller is described in more detail in Section 2.1.

The same basic controller design is incorporated into our part insertion neural controller. This controller focuses on task variability and utilizes force/torque sensations to provide active compliance for assembling parts. Our sample task was to put a peg into a hole that could vary in both its position and orientation. The role of the neural network in this case was to learn the association between the force/torque sensations and the corresponding corrective movements that would result in a successful insertion. This controller is discussed in Section 2.2.

1.5 Experimental Configuration

Our development platform was an industrial PUMA 260 robot, shown in Figure 1, that was controlled directly by a PC-based host. Both autonomous neural network controllers were developed and tested on this platform. The conventional UNIVAL or VAL II controller for the PUMA 260 was not used here; the PC had direct control over the torque signals that were applied to each joint. Optical encoders on each joint provided position feedback information for the PC. A pneumatically-actuated parallel gripper was used to grasp payloads for moving and parts for inserting. For the part insertion neural controller, a six degree-of-freedom (6-dof) force/torque sensor was attached between the robot wrist and the gripper, to sense any contact force during part insertion.

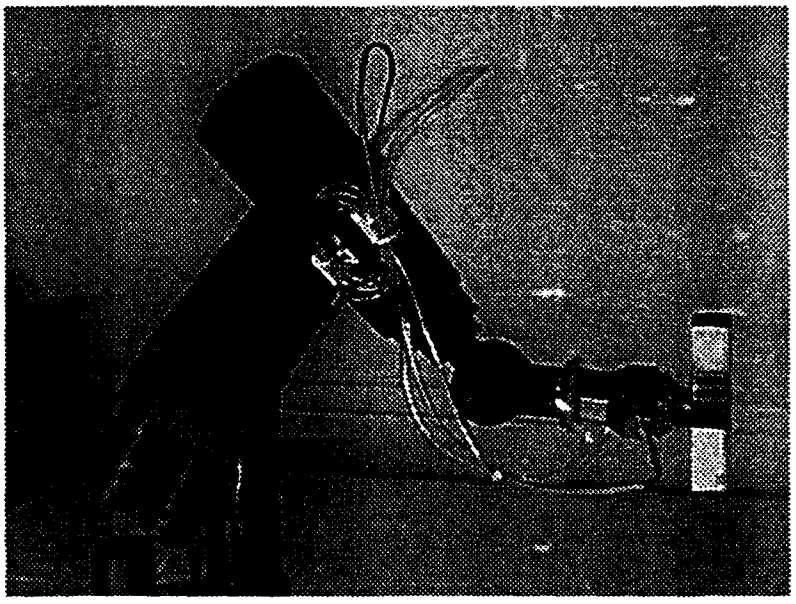


Figure 1: Industrial PUMA 260 robot used for developing both autonomous neural network controllers.

2. Autonomous Neural Network Controllers

2.1 Dynamic Multijoint Neural Controller

2.1.1 Design Description

For a robot with N joints, the dynamic multijoint control module in the neural network controller is composed of N local joint controllers, one of which is shown in Figure 2. Each local joint controller operates in two modes, posture and movement. The role of the posture mode is to keep the arm at its desired position, irrespective of gravity and payload. If a new desired position is set, the posture mode will move the arm there with accuracy but not at the desired speed or with stability. Movement mode is responsible for establishing the speed needed for on-time arrival and the end-point stability. These modes operate in parallel (additively) with the movement mode being activated when the desired position changes.

The local joint controller has four inputs:

1. The initial position of each joint in the arm in joint space (as opposed to physical space). These angles will be denoted by x_{i0}.
2. The desired position for each joint in joint space, denoted by x_{id}.
3. The desired movement time, T_d.

4. A measure of the current payload, P_L. This measure can be subjective and only needs to be repeatable and monotonic with the payload's actual weight.

The first two inputs are used directly by the local joint controller and the second two inputs are used only to compute parameter values.

To perform a movement, an internal estimate of the position and velocity is generated for the i^{th} joint according to

$$\frac{d}{dt}\hat{x}_i(t) = T_{ix}\mathrm{sgn}(x_{id} - x_{i0})u(\hat{x}_i(t)), \tag{1}$$

$$\hat{v}_i(t) = M_{i\alpha}[\hat{x}_i(t) - x_{i0}]\,[x_{id} - \hat{x}_i(t) + M_{i\beta}], \tag{2}$$

where $\hat{x}_i(t)$ is the position estimate, $\hat{v}_i(t)$ is the velocity estimate, and T_{ix} is the position integration timing rate, $M_{i\alpha}$ is the speed gain, and $M_{i\beta}$ is the brake bias. The timing of the movement is

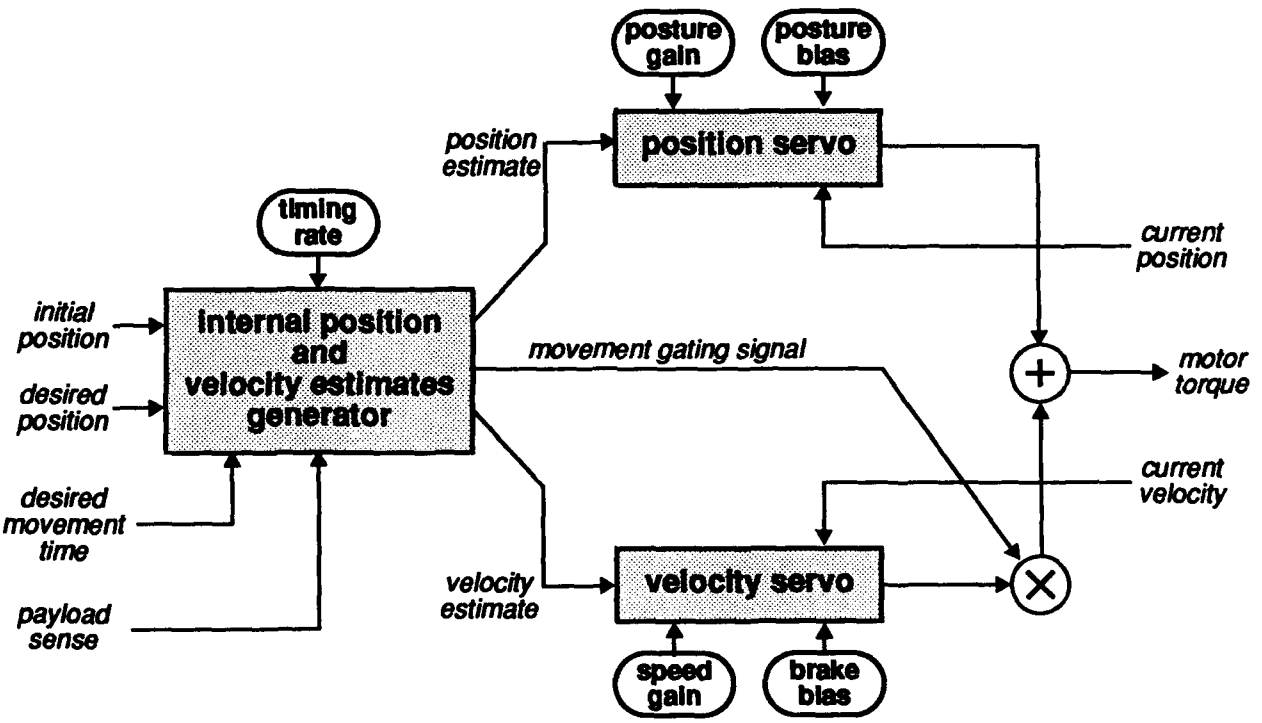


Figure 2: Block diagram of a local joint controller. Parameters in ovals represent neural network outputs. For a robot with N joints, the complete dynamic multijoint control module of the neural network controller consists of N distinct local joint controllers.

governed by T_{ix}, which in turn depends on the desired movement time. The function $u(\hat{x}_i(t))$ is the local movement gating signal defined by

$$u(\hat{x}_i(t)) = \begin{cases} 1 & \text{for } \hat{x}_i(t) \neq x_{id}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

This function is 1 when the movement mode parameters are to be active and 0 when only the posture parameters are in effect.

The true velocity estimate $\frac{d}{dt}\hat{x}_i(t)$ is simply a constant value over some period of time. However, this form does not acknowledge the inertial properties of the arm because it presumes a step change in velocity is possible. The form for $\hat{v}_i(t)$ given above is parabolic and has a symmetrical bell-shaped profile. The peak value of the parabola is determined by the speed gain $M_{i\alpha}$ which is used to adjust the speed of the movement. The peak of $\hat{v}_i(t)$ along $\hat{x}_i(t)$ is set by the brake bias $M_{i\beta}$ which effectively shifts the parabola along the time axis so it can generate either braking or thrusting force at the end of the movement for a smooth stop. This form is more realistic approximation of the velocity because it only presumes a step change in acceleration is possible. Alternatively, a cubic or quartic polynomial can be used to provide smooth acceleration at the expense of increasing the number of parameters.

The position estimate is then used as the reference for a position servo which produces motor torque according to

$$\tau_{i,\,\mathrm{Pos}}(t) = P_{i\alpha}[\hat{x}_i(t) - x_i(t)] + P_{i\beta} \tag{4}$$

where $x_i(t)$ is the current position of the joint, $P_{i\alpha}$ is the position servo gain, and $P_{i\beta}$ is a constant bias to compensate for external forces such as gravity. This torque term constitutes the posture mode. In a similar formulation, the movement mode is composed of a velocity servo that is driven by the velocity estimate according to

$$\tau_{i,\,\mathrm{Mov}}(t) = V_{i\alpha}[\hat{v}_i(t) - v_i(t)], \tag{5}$$

where $v_i(t)$ is the current velocity of the joint and $V_{i\alpha}$ is the velocity servo gain.

The total torque signal sent to the motor amplifier, $\tau_i(t)$, is the sum of the torques produced by the position and velocity servos, with the movement mode term gated by the movement signal

function:

$$\tau_i(t) = \tau_{i,\,\mathrm{Pos}}(t) + [\tau_{i,\,\mathrm{Mov}}(t) \cdot u(\hat{x}_i(t))] \,. \tag{6}$$

The role of each term has physical significance. The position servo maintains movement timing and final position accuracy. The velocity servo provides movement stability by thrusting and braking during the movement to compensate for inertial and dynamic coupling forces.

At the end of the movement phase (signaled by $u(\hat{x}_i(t))$ going from 1 to 0), the performance of the movement is observed by each local joint controller. Three measurements are taken:

1. The time when the movement phase ended, denoted by T_i.
2. The position of the joint, x_{im}.
3. The velocity of the joint, v_{im}.

If the movement had some instabilities, the arm may still be in motion for some small amount of time after the movement phase ended. When the arm finally stops moving (under the sole influence of the posture mode position servo), a fourth measurement is available: the final position of the joint, x_{if}. These measurements translate into the following error quantities for each local joint controller:

1. The arrival time error, $\Delta T_i = T_d - T_i$.
2. The movement position error, $\Delta x_{im} = x_{id} - x_{im}$.
3. The movement velocity error, $\Delta v_{im} = v_{id} - v_{im}$ which equals $-v_{im}$ since v_{id}, the desired velocity at the end of the movement, is 0.
4. The posture position error, $\Delta x_{if} = x_{id} - x_{if}$.

These errors can be used to adapt the parameters of the local joint controller.

In the most general form, the adaptable parameters are T_{ix}, $P_{i\alpha}$, $P_{i\beta}$, $V_{i\alpha}$, $M_{i\alpha}$, and $M_{i\beta}$. During our experiments, both $P_{i\alpha}$ and $V_{i\alpha}$ were held constant for simplicity so no error functions were explored. The error functions for the remaining parameters are:

$$\delta_{T_{ix}} = \Delta T_i, \tag{7}$$

$$\delta_{P_{i\beta}} = \Delta x_{if}, \tag{8}$$

$$\delta_{M_{i\alpha}} = -\Delta v_{im}, \tag{9}$$

and

$$\delta_{M_{i\beta}} = \Delta x_{im}. \tag{10}$$

Each adaptable parameter is encoded by a separate neural network. These error functions are used to update the weights in their respective networks. The basic network model used by all parameters is an adaptive topographical map that is excited by a fixed bell-shaped activation function centered at the inputs to the map. The output of the map is the sum of the weighted outputs of the map. This structure is best illustrated using an example.

The position estimate integration rate (T_{ix}) depends only on the desired movement time (T_d) and the initial (x_{i0}) and final (x_{id}) positions of the movement. Since these three inputs are independent, the corresponding map for T_{ix} needs to have three dimensions. Let W represent the map so W_{jkl} denotes the neural weight at $x_{i0} = j$, $x_{id} = k$, and $T_d = l$. Furthermore, let the bell-shaped activation function be the three-dimensional Gaussian distribution,

$$g(j, k, l) = C \exp\left[-\frac{(j-x_{i0})^2 + (k-x_{id})^2 + (l-T_d)^2}{2\sigma^2}\right] \tag{11}$$

where C is a normalization constant and σ is the half-width. The map output is computed using

$$T_{ix} = \sum_j \sum_k \sum_l W_{jkl}\, g(j, k, l). \tag{12}$$

Given the error signal $\delta_{T_{ix}}$, the map is adapted using

$$\Delta W_{jkl} = \eta \delta T_{ix} g(j, k, l) \tag{13}$$

where ΔW_{jkl} is the change in the weight at index position (j, k, l) and η is the learning rate.

The maps for the other parameters are slightly more complex in that there are more inputs. For example, the posture parameter $P_{i\beta}$ depends upon the current payload P_L and the final positions of *all* joints. The movement parameters $M_{i\alpha}$ and $M_{i\beta}$ depend upon the movement time, the current payload, and the initial and final positions of all joints. In general, the maps for parameters like $M_{i\alpha}$

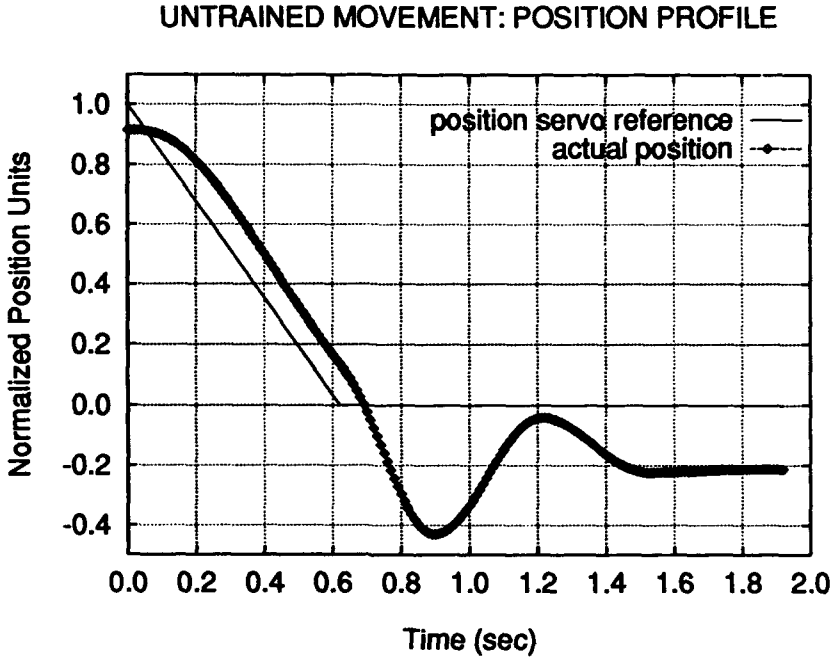
and $M_{i\beta}$ could $(N+2)$-dimensional. However, it is not necessary to relate the initial position of joint i (x_{i0}) with the final position of joint n (x_{nd}). It is important, though, to relate the movement of joint i (x_{i0}, x_{id}) with the movement of joint n (x_{n0}, x_{nd}). Thus, the $(N+2)$-dimensional maps can be reduced to a set of N two-dimensional maps that relate x_{i0} and x_{id} for each joint and then have a separate two-dimensional map for the time and payload inputs. The output of all these maps can be summed together to form the desired parameter, such as $M_{i\alpha}$. The error value $\delta_{M_{i\alpha}}$ is then applied to all the component maps in the normal way.

2.1.2 Performance Results

For simplicity in our experiments, only two of the six PUMA 260 joints, the shoulder J2 and the elbow J3, were under the control of the dynamic multijoint neural controller. The remaining four joints were simply held at their position via a conventional PID (proportional-integral-derivative) controller. This simplification is valid because these two joints retain all the inertial, coupling, and gravitational forces that the complete six-joint controller would have to face.

The performance of the dynamic multijoint neural controller is illustrated by the corresponding video tape that accompanies this report. In this demonstration, the controller learns to move between four positions in about 25 trials per movement. At the joint level, the movements were accurate to within 0.1% of the range of each joint. Furthermore, the final velocity was within 1% of the maximum desired joint velocity. This performance exceeds that of conventional dynamic control methods but unfortunately is comparable to recently developed commercially-available algorithms that rely on detailed kinematic and dynamic models of the robot [see, for example, Hanafusa and Hirochika (1985) and Whitcomb *et al.* (1993)].

Sample movement profiles for the movements in the video tape are shown in Figures 3 and 4 for the shoulder joint. In Figure 3, the position profile as a function of time is plotted for both before and after training. This profile consists of the position estimate $\hat{x}_1(t)$, which serves as the position servo reference signal, and the actual position $x_1(t)$. The corresponding velocity profiles depicting $\hat{v}_1(t)$, the velocity servo reference signal, and $v_1(t)$, the actual velocity, are shown in Figure 4. The input parameters for this movement are: $x_{10} = 1, x_{1d} = 0, T_d = 0.6$, and $P_L = 0$. Note that the untrained controller exhibits significant end-point instability by the oscillations at the end of the movement. The sharp change in velocity for this case, shown in the top curve of Figure 4, is due to the movement gating signal $u(\hat{x}_1(t))$ turning off. Since gravity is present, the final position for



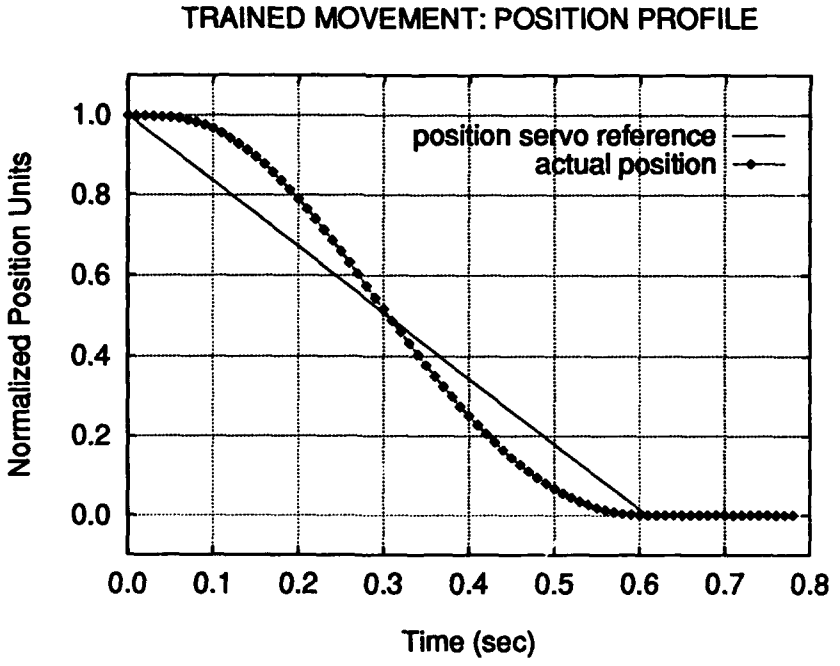
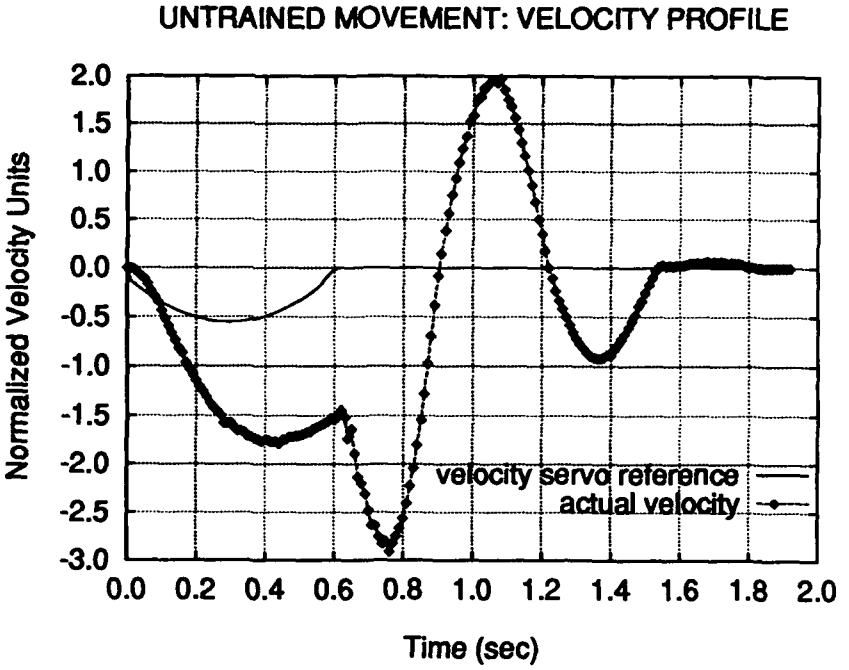


Figure 3: Sample movement profiles showing the position as a function of time for both the untrained and trained movement. These profiles are taken from one of the movements made by the shoulder joint of the robot in the video tape. The corresponding velocity profiles are shown in Figure 4.



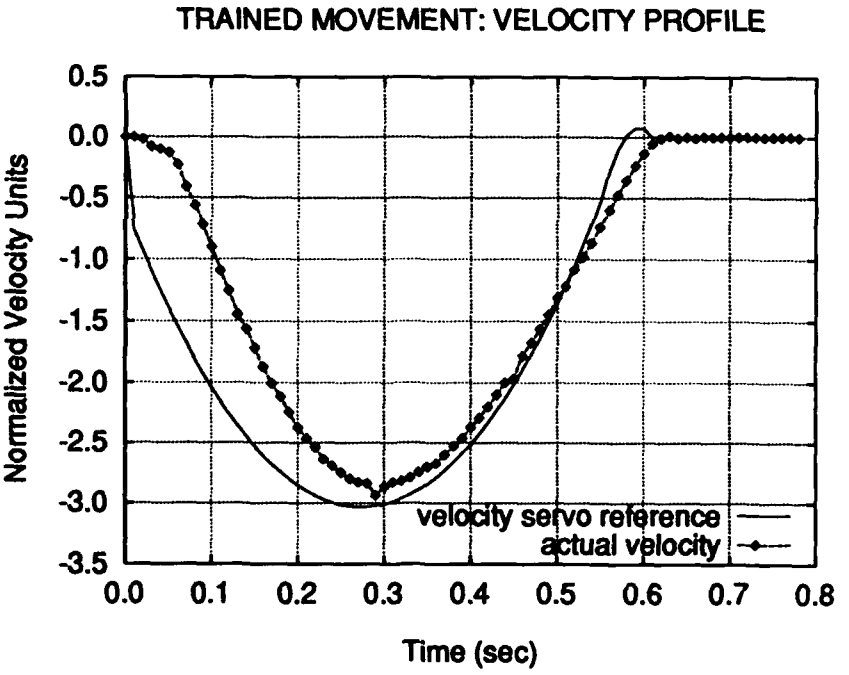


Figure 4: Velocity profiles corresponding to the movement profiles shown in Figure 3. Note that the significant end-point oscillations in the untrained movement have been eliminated after the dynamic multijoint neural controller has been trained.

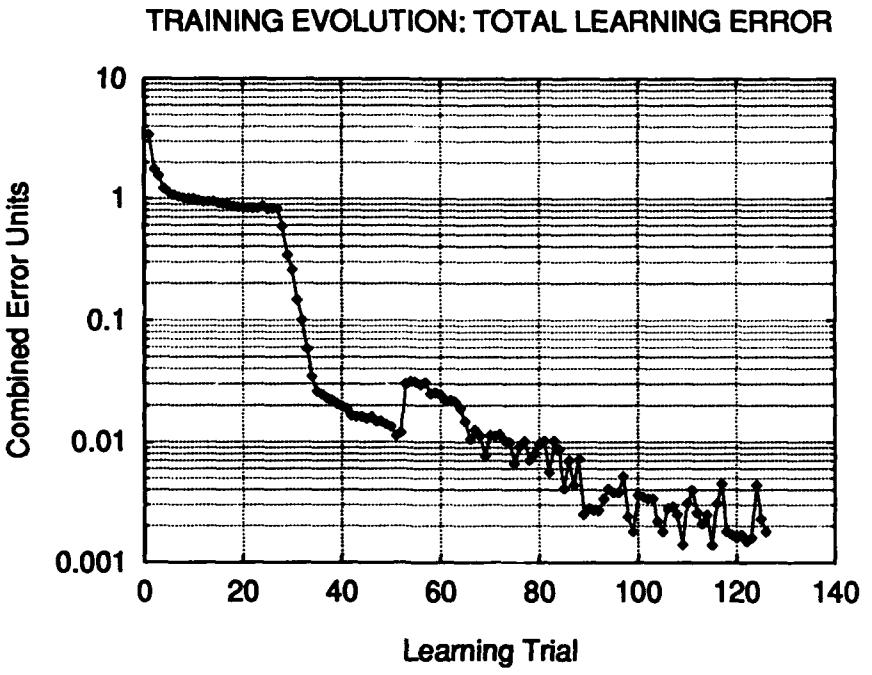


Figure 5: Evolution of the total learning error in which different movement parameters are adapted in stages to illustrate the learning process.

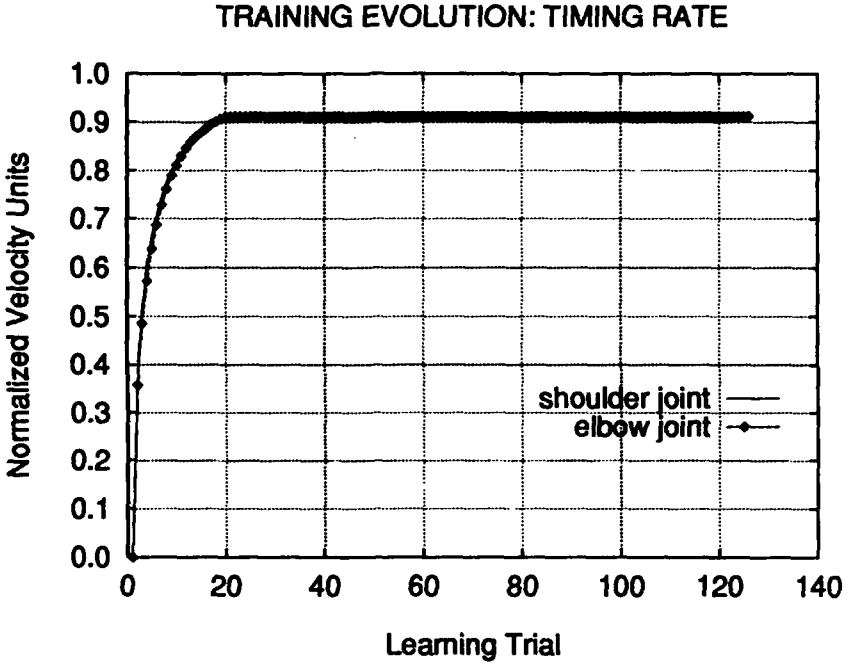
the untrained movement is below the desired position. After training, the controller produces smooth, stable, and accurate movements.

The training process is illustrated in Figures 5 through 9. Although all parameters can be adapted concurrently, the training process depicted in these curves adapts different parameters in stages to study how performance improves with learning. Figure 5 shows the evolution of the total learning error, defined by

$$\delta_{\text{total}} = \sum_i (\delta_{T_{ix}} + \delta_{P_{i\beta}} + \delta_{M_{i\alpha}} + \delta_{M_{i\beta}}) \tag{14}$$

at each trial, during this adaptation. The level or shallow-sloped regions of the total learning error curve indicate that the parameters being adapted in this stage are converging to steady-state values.

In the first learning stage, only the timing rate T_{ix} and the posture bias $P_{i\beta}$ for both joints are allowed to adapt. They remain adaptable during the entire learning process. As shown in Figures 6 and 7, these parameters rapidly converge in about 20 trials. Note that during these trials, the



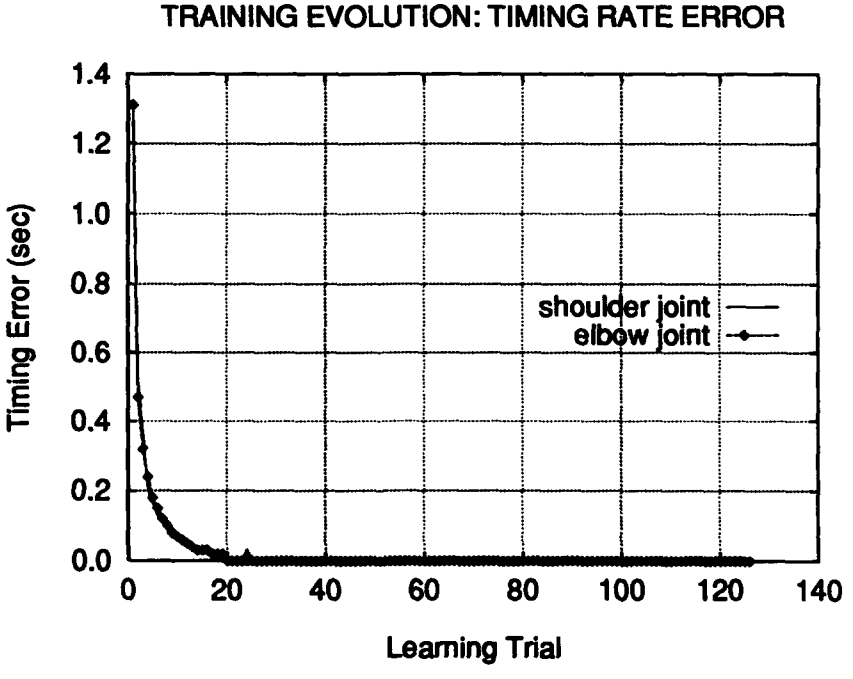


Figure 6: Adaptation evolution of the timing rate T_{ix} for both joints. The upper plot shows the T_{ix} values and the lower plot shows the learning error $\delta_{T_{ix}}$.

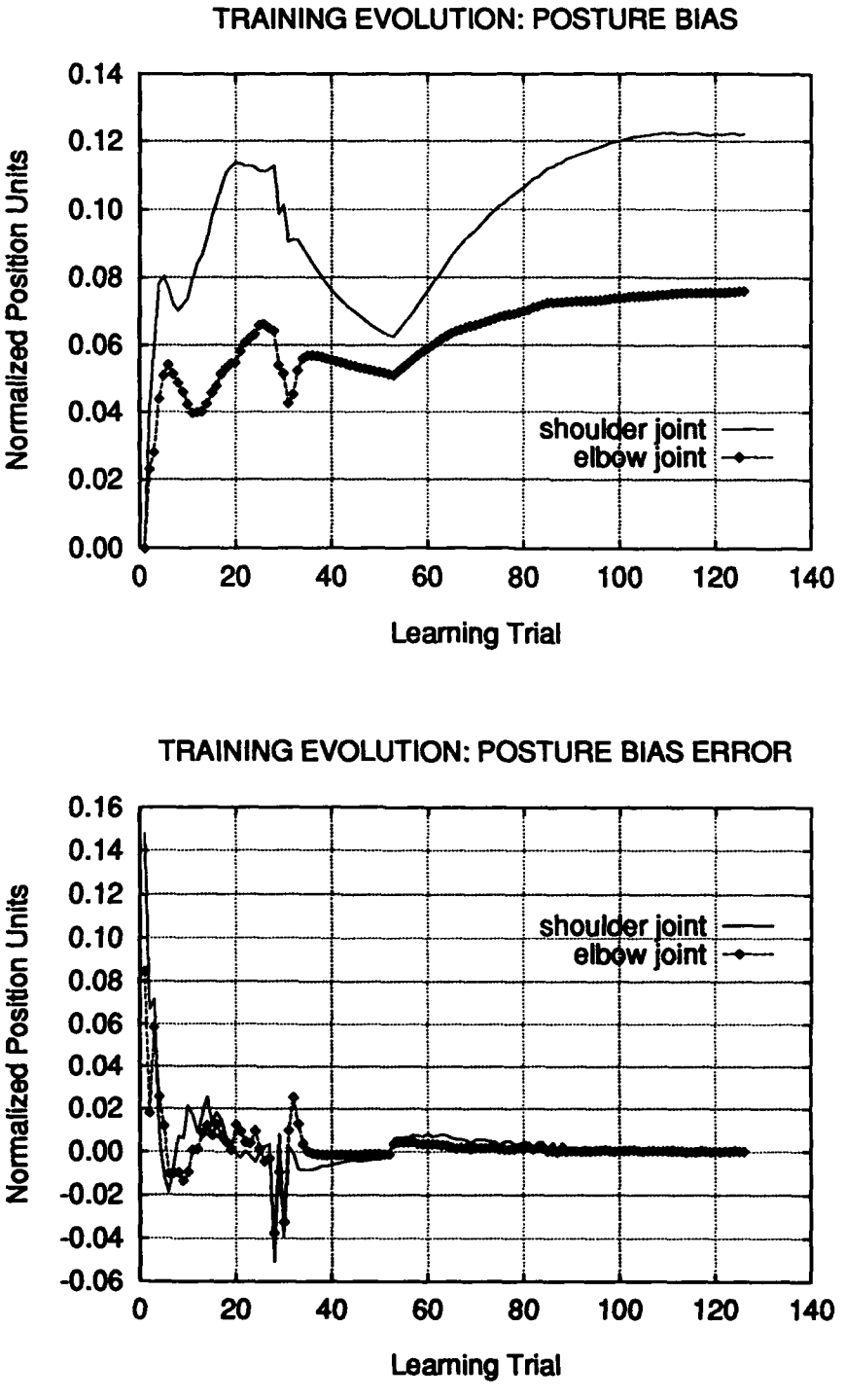
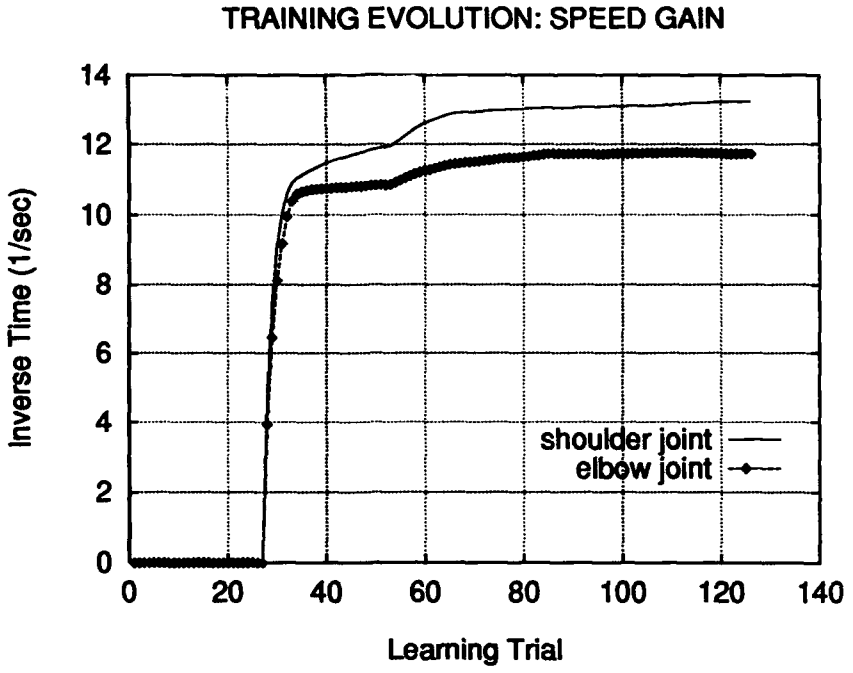


Figure 7: Adaptation evolution of the posture bias $P_{i\beta}$ for both joints. The upper plot shows the $P_{i\beta}$ values and the lower plot shows the learning error $\delta_{P_{i\beta}}$.



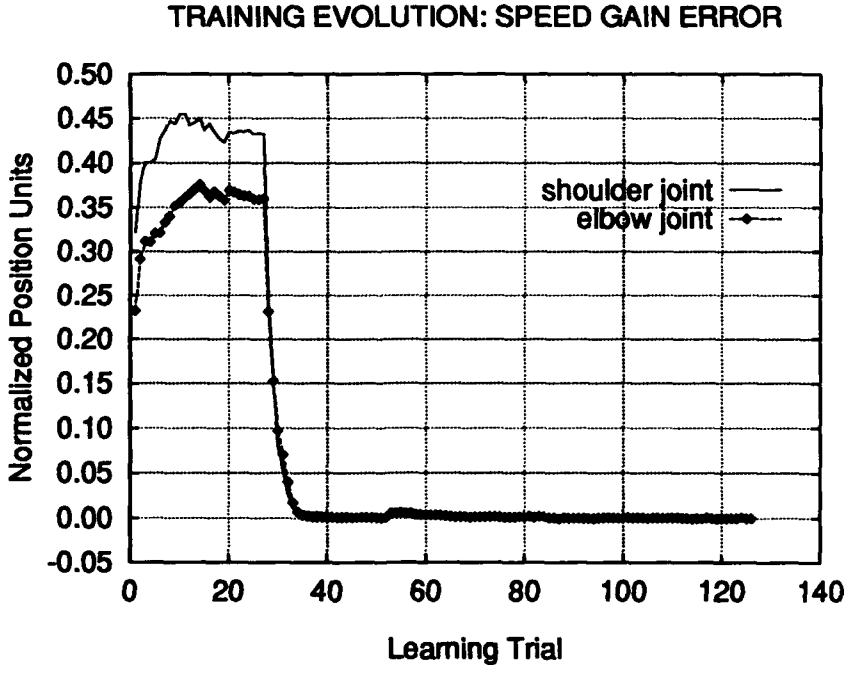


Figure 8: Adaptation evolution of the speed gain $M_{i\alpha}$ for both joints. The upper plot shows the $M_{i\alpha}$ values and the lower plot shows the learning error $\delta_{M_{i\alpha}}$.

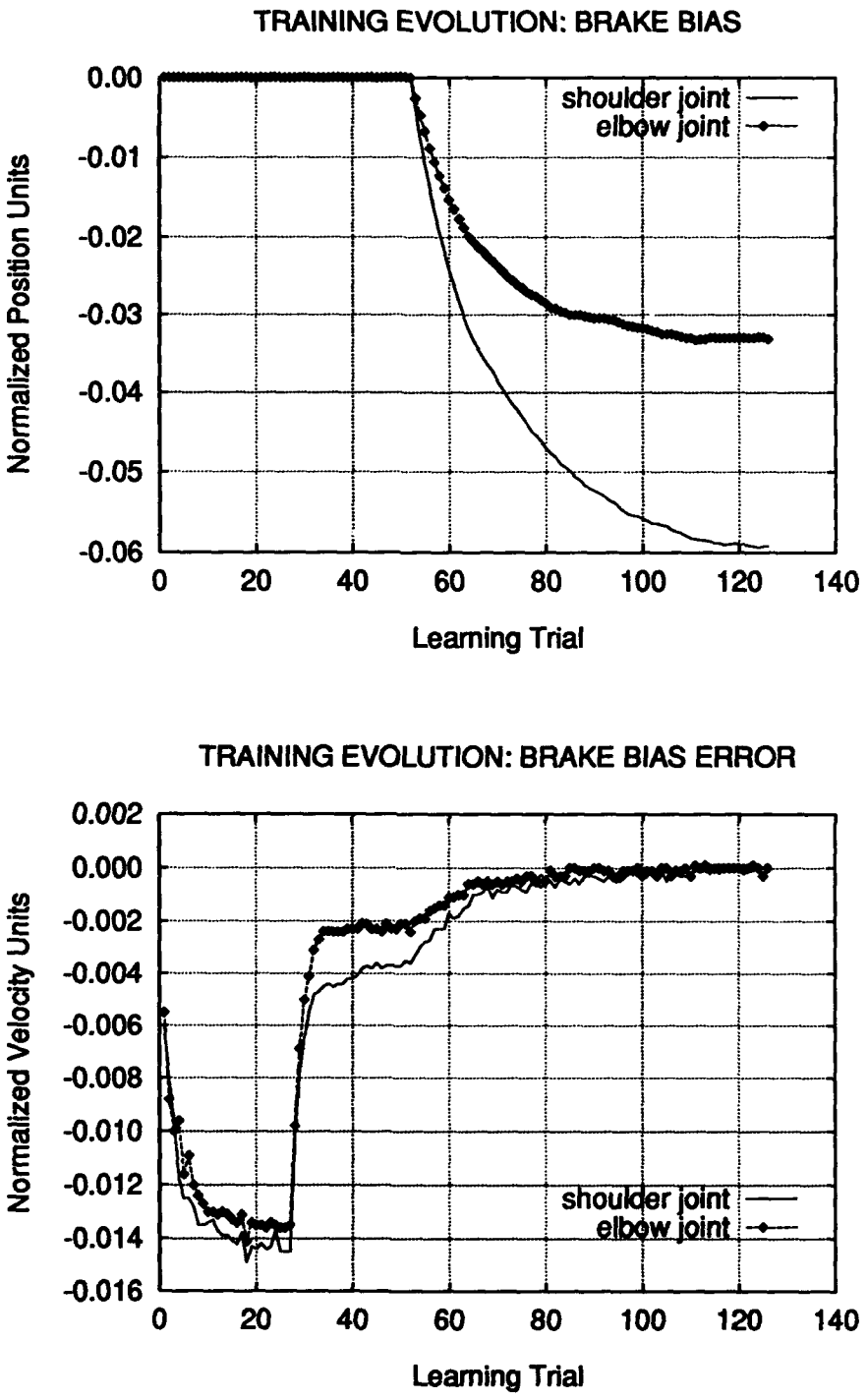


Figure 9: Adaptation evolution of the brake bias $M_{i\beta}$ for both joints. The upper plot shows the $M_{i\beta}$ values and the lower plot shows the learning error $\delta_{M_{i\beta}}$.

learning errors for the speed gain and the brake bias are relative large and grow slightly as the timing rate and posture bias converge. This growth is due to the timing rate increasing, which results in a faster and more unstable movement.

As shown in Figure 8, the speed gain $M_{i\alpha}$ is allowed to adapt after trial 27 in the next learning stage. The speed gain converges rapidly initially and quickly slows to more gradual changes. This causes the movement to become more smooth and have velocity profile that is more bell-shaped. As a result, the brake bias error decreases, as shown in Figure 9. However, because the movement is faster, the joints overshoot more than the previous stage. Because of friction, the joints now stop moving past the desired position, so the posture bias must decrease to compensate (Figure 7).

In the final learning stage, the adaptation of the brake bias $M_{i\beta}$ is enabled after trial 52. The brake bias gradual builds up (Figure 9), reducing the overshoot. The speed gain needs to increase slightly to make the joints reach their desired positions now that additional braking is available (Figure 8). Also, the posture bias readjusts to commodate the changes in the speed gain and brake bias (Figure 7). In both the second and third learning stages, the timing rate remains constant because its error condition does not depend upon the position or velocity of the joints. At the end of the adaptation, the movements are smooth, stable, and accurate, just like the profiles shown in the lower plots of Figures 3 and 4.

This three-stage learning process could be done in one stage by adapting all parameters in parallel. Larger learning rates selected independently for each parameter could also decrease convergence time. However, if the learning rates become too large, the adaptation can become unstable, thus preventing the parameters from converging.

2.2 Part Insertion Neural Controller

2.2.1 Design Description

The part insertion process involves two control phases, first finding the hole (the *find-hole* phase) and then guiding the part into the hole (the *guide-part* phase). Two phases are needed because the meaning of the same force/torque feedback can be different between them. In order to provide active compliance for both phases, the part insertion neural controller must learn to generate corrective movements given sets of force/torque sensations and the current control phase. Furthermore, the controller must be able to switch between phases so that the part can be inserted smoothly and continuously.

A block diagram of the part insertion neural controller is shown in Figure 10. It is composed of five

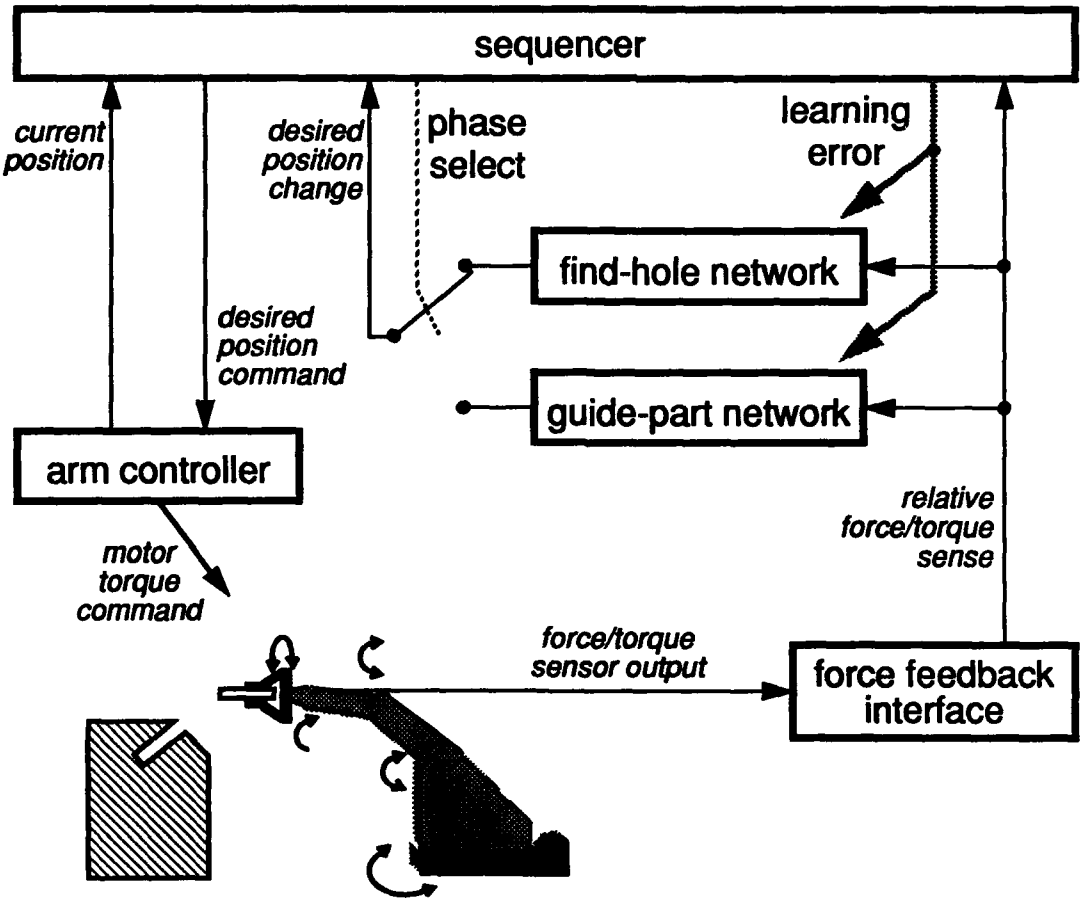


Figure 10: Block diagram of the part insertion neural controller.

components: a sequencer, an arm controller, an interface for the force/torque sensor, and two mapping neural networks, one for each control phase. The sequencer coordinates the operation of all the components and generates the necessary movement commands in terms of the desired joint positions $X_d(t)$ for all joints. The arm controller moves the arm to the most recent commanded position by computing the required motor torque signals $\tau(t)$ for all joints. Our dynamic multijoint neural controller could be used as the arm controller here. The force/torque feedback interface converts the 6 absolute force and torque values $F(t)$ from the force/torque sensor into relative deviations from the expected force/torque values $\Delta F(t)$. Both neural networks store a mapping from $\Delta F(t)$ to the appropriate corrective movement expressed as a relative change in the desired

position for each joint, $\Delta X_d(t)$. However, only one network can be selected by the sequencer at a time. When the find-hole network is active, the relative position change is used to find the hole opening. Similarly, the $\Delta X_d(t)$ from the guide-part network is used to guide the part into the hole when that phase is active.

The part insertion neural controller has two modes, learning and performance. The function of the learning mode is to train the neural networks using the controller's experience with actual force/ torque signals. Performance mode is simply the part insertion process. Both modes must accommodate the two control phases. The learning mode for both phases is structurally similar but differs in the sequencer operation. For a smooth insertion, the performance mode must provide a mechanism for switching between the control phases.

To train the guide-part network, the initial learning experience for the controller cannot come from inserting a part into its hole since the hole location is not precisely known. Instead, the learning experience comes from simply reversing the insertion process. First, the part is manually placed into its destination hole and the robot arm is moved so the part can be gripped. Let the positions of the arm joints at this starting position be denoted by X_0. Then the sequencer explores how to remove the part by generating random but nearby position commands $X_d(t)$. As the arm controller moves the arm with the part held by the gripper, the force/torque feedback interface computes the force/torque deviation $\Delta F(t)$ with respect to the initial force/torque values (the expected values). When the magnitude of the deviation vector $|\Delta F(t)|$ exceeds a threshold ΔF_{max}, the sequencer commands the arm to stop at its current position and the guide-part network is trained to associate the current force/torque deviation $\Delta F(t)$ with the relative position vector which will decrease the force/torque deviation. This vector is the difference between the current reference position X_0 and the actual position vector $X(t)$ when the force threshold was reached. The learning error for the network is thus

$$\delta(t) = X_0 - X(t) \tag{15}$$

when the event

$$|\Delta F(t)| \geq \Delta F_{max} \tag{16}$$

first occurs. When the arm reaches an exploratory position $X_d(t)$ with a minimum value for $|\Delta F(t)|$, this position is set to be the new reference position X_0. By iterating this procedure, the reference

position X_0 is slowly incremented in such a way that the part is removed from its hole. Furthermore, by exploring several nearby positions $X_d(t)$, the guide-part network can be trained in one pass at pulling the part out of its hole.

Training the find-hole network follows a different procedure. With the part being grasped by the gripper, the arm is positioned so that the part is in the correct position to enter the hole. This position is set to be the reference position X_0 for this training. The sequencer then commands the arm to move in random directions that are toward the hole. When the force/torque deviation magnitude $|\Delta F(t)|$ reaches the threshold ΔF_{max}, the find-hole network is trained to associate $\Delta F(t)$ with the current relative position that will return the arm to X_0 for another attempt at finding the hole. The learning error for this case is

$$\delta(t) = X_0 - X(t) \tag{17}$$

when the event

$$|\Delta F(t)| \geq \Delta F_{max} \tag{18}$$

first occurs, the same error condition that is used to train the guide-part network.

In performance mode, the sequencer starts in the find-hole control phase and simply commands the arm (with the part in its gripper) to move along a preprogrammed path to the expected position for the hole. If the hole has moved slightly, the part will make contact with the edge of the hole, thereby generating a nonzero force/torque deviation signal $\Delta F(t)$. When $|\Delta F(t)| \geq \Delta F_{max}$, the find-hole network is accessed to produce the relative position offset $\Delta X_d(t)$. The arm is commanded to move to the absolute position

$$X_d(t+1) = X(t) + \Delta X_d(t) \tag{19}$$

for the next attempt at inserting the part into the hole. This process is repeated until the part enters the hole. With only a force/torque sensor available and no kinematic model of the robot, this determination can be made by monitoring one or more joint positions for crossing preset values. This condition imposes one limit on the range of variability on the hole position.

2.2.2 Performance Results

The performance of the part insertion neural controller is demonstrated by the corresponding video tape that accompanies this report. In the demonstrations, the test part is a cylindrical peg with a

rounded tip. It has a diameter of 0.97 inch and a length of 6 inches, 2 inches of which are to be inserted into a 1.06 inch diameter hole. The resulting insertion tolerance is about 0.045 inch or about 5% of the hole diameter. At the midpoint of the peg are two slots so that the peg could be grasped reliably by a parallel-finger gripper. In our test experiments, the position of the centerpoint of the hole opening could vary up to about one quarter of the hole's diameter (approximately 0.25 inch). Furthermore, the angular orientation of the hole axis could deviate up to $\pm 10°$. The demonstrations show the part insertion neural controller successfully controlling the PUMA 260 robot inserting the peg into the hole under various hole placements and orientations. Since it is not a complete prototype yet, the specifications on the range of active compliance it offers are not available. We plan on using the video tape as marketing tool for pursuing prospective customers of this technology.

2.2.3 Remaining Problems

By its nature, the force/torque sense is a local feedback mechanism that is usable only when the part is in contact with the hole. If the hole location is moved too far away from its expected position, the force/torque signals cannot be used during the find-hole phase because they no longer are unique. A more global sensing mechanism is needed such as a vision system. The vision system can also be used to determine when the part actually enters the hole much more reliably than using any preset joint positions.

Another problem that can arise by using solely force/torque senses to find the hole opening is that the same force/torque signals can arise in two different situations, each of which requires a different corrective movement. An example of this condition is illustrated in Figure 11. In both situations, the location of the hole is shifted slightly to the left. However, the hole orientations are opposite. The sensed force/torque signals indicate that the lower portion of the part is being pushed to the left slightly. The desired corrective movements, though, are in opposite directions to make the part have the proper orientation. For small angular variations in the orientation, the guide-part mode can handle this case once the part is in the hole more deeply. However, for large angular orientations, the part could become jammed and prevent a successful insertion. As a temporary solution to this problem, we use two force/torque readings obtained from trying to insert the part using two known adjacent starting points. This double sense provides contextual information about the location and orientation of the hole. In effect, it is a simple first-order approximation of the spatial gradient for entering the hole. A better way to resolve this problem is to use a vision system to incorporate a sense of the orientation of the hole opening and allow this sense to be associated with the proper orientation for the part. This training could be accomplished using a self-consistent learning

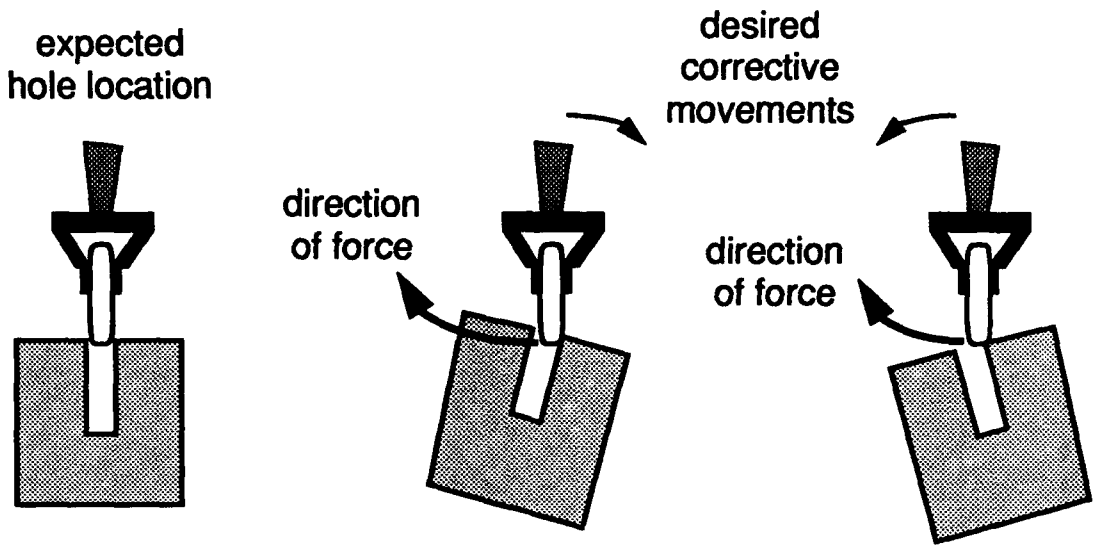


Figure 11: Example situation when the force/torque signals are the same but the desired corrective movement is different.

algorithm such as Kuperstein's INFANT model (1988, 1991).

Alternatively, this ambiguity problem could be solved by introducing joint-level active compliance into the arm controller. In this case, any change in a joint's position due to a mismatch in the part's orientation could be compensated for by changing the target position for that joint to be its current position. Instead of using a position discrepancy, joint-level active compliance can also be sensed by using individual force sensors on each joint, such as Hall-effect current sensors for each motor. Given enough sensitivity on the individual joint force sensors, the force/torque sensor at the gripper may not even be needed.

3. Commercialization

Symbus Technology has a number of criteria we use to evaluate the commercial potential for bringing a product to market:

1. The benefit that the product offers should fill a large need and have substantial value.
2. The product should be profitable, which means that the value the customer perceives of the product is higher than the cost to build it.

3. There should not be many competitors for the benefit that the product offers the customer.
4. Product development should be financed from available sources.

Using these criteria we talked to informed customer prospects in the automation market to get some market feedback about our product concept in part insertion. We made a video tape of our part insertion demonstration and mailed it to the following companies:

1. ABB Robotics - robot manufacturer.
2. Advanced Robotics Research - robot reseller and controller manufacturer.
3. JR3 - force-torque sensor manufacturer.
4. ATI - force-torque sensor manufacturer.
5. Allen Bradley Co. - industrial control manufacturer.
6. Precision Robots Inc. - robot manufacturer.
7. Harbor Research - robot market consultant.
8. Trellis - robot software manufacturer.

From the companies that we talked to, we got the following feedback: Our technology fills the need of automatically inserting parts in a highly variable environments. The benefits we are offering include increasing the scope of variability beyond currently available methods in existing applications and creating new applications that can only be accomplished by accommodating higher variability of misalignment than is possible today. Today's possible work-arounds include jiggling the part into place and allowing a passive compliant device to accommodate the misalignment between the part and the insertion. The largest market segments that can use our benefit include the military which has special requirements for loading ammunition and the service market because the interactions between robots and the environment are variable and not easily constrained.

There are already a number of obstacles to market. One obstacle is creating a new market which requires expensive missionary work to educate possible customers. Another obstacle is that the product's capability requires enabling technology that is not commonly available. This creates delays to product distribution. In our case of adaptive part insertion in the service market, the enabling technology is a more developed infrastructure of associated adaptive functions for moving, reaching, picking, placing, tracking and catching. For the robot service market to grow

large, all of these functions in addition to part insertion need to interact and work together.

Since our product will be software, the cost of manufacturing the product is low, but the development costs are high. Once developed, the product would have high profit margins. We have no knowledge of commercial competitors for the benefit we are offering.

We will continue to contact more companies, especially in the military, robot system integration and service robot markets. Our next step is to get a clear view of the scope and size of the part insertion market and then develop commercial relationships with companies that will help us in the appropriate distribution channel in the market.

4. Summary and Conclusions

We have developed two autonomous neural network controllers for allowing industrial robots to operate in highly variable environments. These controllers address variability at both the movement and task levels. They learn appropriate performance by experience. In addition, they can learn continuously so that the controllers can adapt to slowly varying changes in the robot and operating environment.

For movement-level variability, our dynamic multijoint neural controller can learn to move variable payloads from one point to another with a high degree of accuracy and end-point stability without any knowledge of the robot's kinematics or dynamics. It only requires the number of joints and their movement range. The performance of the controller is comparable to recently-developed commercially-available controllers. As a result, this controller will not be pursued as a product to be sold to robot controller manufacturers because it does not offer enough benefits for them to switch control methodologies. However, the dynamic multijoint neural controller may be useful for other process control applications. Since it is model independent, it could be applied to inertial control problems such as temperature control and fluid control.

For task-level variability, our part insertion neural controller demonstrates the feasibility of having an autonomous controller learn to use force/torque feedback to guide a part insertion task with unconstrained alignment. In this case, the location and orientation for the part destination can vary over consecutive trials. This capability can give existing robots a much larger range of compliance than existing methods (jiggling the part or using remote center of compliance fixtures to hold the part). For wider variations in the hole location, a more global sensing system such as vision could be incorporated into the part insertion neural controller.

The technology demonstrated by our autonomous neural controllers should be very beneficial for

robot controller manufacturers. It can increase the functionality of existing robots and expand the range of robot applications. Furthermore, as more robots are used in unconstrained environments such as in the area of service robotics, our technology will be crucial for making the robots more productive and efficient.

Symbus has begun it initial market research effort in determining the size and scope of the potential market in active compliance for part insertion. Initial market feedback on a video tape demonstration of our capabilities is indeterminate. We will continue to contact more companies, especially in the military, robot system integration and service robot markets.

References

Hanafusa, H., and Hirochika, I. (1985). Robotics Research, The Second International Symposium.

Kuperstein, M. (1988). Neural network model for adaptive hand-eye coordination for single postures. *Science*, **239**, 1308-1311.

Kuperstein, M. (1991). INFANT neural controller for adaptive sensory-motor coordination. *Neural Networks*, **4**, 131-145.

Whitcomb, L. L., Rizzi, A. A., and Koditschek, D. E. (1993). Comparative experiments with a new adaptive controller for robot arms. *IEEE Trans. on Robotics and Automation*, **9**, 59-70.